```
/*
  Vcc - A supply voltage measuring library for Arduino

  Created by Ivo Pullens, Emmission, 2014

  Inspired by:
  http://provideyourown.com/2012/secret-arduino-voltmeter-measure-battery-voltage/

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
  Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/

#include "Vcc.h"
#include "EMA.h"

Vcc::Vcc( const float correction )
  : m_correction(correction)
{
}

#if defined(__AVR_ATmega32U4__) || defined(__AVR_ATmega1280__) || defined(__AVR_ATmega2560__)
  #define ADMUX_VCCWRT1V1 (_BV(REFS0) | _BV(MUX4) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1))
  #define _IVREF 1.1
  #define _IVREF 1100L
  #define _ADCMAXRES 1024.0
  #define _ADCMAXRES 1024L
#elif defined (__AVR_ATtiny24__) || defined(__AVR_ATtiny44__) || defined(__AVR_ATtiny84__)
  #define ADMUX_VCCWRT1V1 (_BV(MUX5) | _BV(MUX0))
  #define _IVREF 1.1
  #define _IVREF 1100L
  #define _ADCMAXRES 1024.0
  #define _ADCMAXRES 1024L
  #elif defined (__AVR_ATtiny25__) || defined(__AVR_ATtiny45__) || defined(__AVR_ATtiny85__)
  #define ADMUX_VCCWRT1V1 (_BV(MUX3) | _BV(MUX2))
  #define _IVREF 1.1
  #define _IVREF 1100L
  #define _ADCMAXRES 1024.0
  #define _ADCMAXRES 1024L
#elif defined(__LGT8FX8P__)
  #define ADMUX_VCCWRT1V1 (_BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX0))
  #define _IVREF 1.024
  #define _IVREF_FAST 1024L
  #define _ADCMAXRES 4096.0
  #define _ADCMAXRES_FAST 4096L
#elif defined(__LGT8FX8E__)
  #define ADMUX_VCCWRT1V1 (_BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1))
```

```
  #define _IVREF 1.25
  #define _IVREF 1250L
  #define _ADCMAXRES 4096.0
  #define _ADCMAXRES 4096L
#else // defined(__AVR_ATmega328P__)
  #define ADMUX_VCCWRT1V1 (_BV(REFS0) | _BV(MUX3) | _BV(MUX2) | _BV(MUX1))
  #define _IVREF 1.1
  #define _IVREF_FAST 1100L
  #define _ADCMAXRES 1024.0
  #define _ADCMAXRES_FAST 1024L
#endif

uint16_t adcRead_(void){
  ADCSRA |= _BV(ADSC);
  while (bit_is_set(ADCSRA, ADSC));
  return ADC;
}

uint16_t Read_(void)
{
  analogReference(DEFAULT);    // Set AD reference to VCC
#if defined(__LGT8FX8P__)
  ADCSRD |= _BV(BGEN);         // IVSEL enable
#endif
  // Read 1.1V/1.024V/1.25V reference against AVcc
  // set the reference to Vcc and the measurement to the internal 1.1V reference
  if (ADMUX != ADMUX_VCCWRT1V1)
  {
    ADMUX = ADMUX_VCCWRT1V1;
    // Wait for Vref to settle. Bandgap reference start-up time: max 70us
    delayMicroseconds(350);
  }

  uint16_t pVal;
  uint16_t pVal_filtered;
  static EMA<2> EMA_filter;

#if defined(__LGT8FX8P__)
  uint16_t nVal;
  ADCSRC |=  _BV(SPN);
  nVal = adcRead_();
  ADCSRC &= ~_BV(SPN);
#endif

  pVal = adcRead_();

#if defined(__LGT8FX8P__)
  pVal = (pVal + nVal) >> 1;
#endif

// Logicgreen gain-error correction
#if defined(__LGT8FX8E__)
  pVal -= (pVal >> 5);
#elif defined(__LGT8FX8P__)
  pVal -= (pVal >> 7);
#endif

  pVal_filtered = EMA_filter(pVal);

  return pVal_filtered;
}
```

```
119
120 float Vcc::Read_Volts(void)
121 {
122   uint16_t pVal_filtered;
123
124   pVal_filtered = Read_();
125
126   // Calculate Vcc (in V)
127   float vcc = m_correction * _IVREF * _ADCMAXRES / pVal_filtered;
128
129   return vcc;
130 } // end Read_Volts
131
132 uint16_t Vcc::Read_Volts_fast(void)
133 {
134   uint16_t pVal_filtered;
135
136   pVal_filtered = Read_();
137
138   // Calculate Vcc (in mV)
139   unsigned long vcc = _IVREF_FAST * _ADCMAXRES_FAST / pVal_filtered;
140   //Serial.println(vcc);
141   return (uint16_t)vcc;
142 } // end Read_Volts_fast
143
144 float Vcc::Read_Perc(const float range_min, const float range_max, const boolean
clip)
145 {
146   // Read Vcc and convert to percentage
147   float perc = 100.0 * (Read_Volts()-range_min) / (range_max-range_min);
148   // Clip to [0..100]% range, when requested.
149   if (clip)
150     perc = constrain(perc, 0.0, 100.0);
151
152   return perc;
153 }
154
```